SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 9, 281–297 (1997)

Research

A Quantitative Comparison of Perfective and Corrective Software Maintenance

JOEL E. HENRY[1]* AND JAMES P. CAIN[2]

[1]*East Tennessee State University, Johnson City, TN 37614, U.S.A.*
[2]*Integrated System Consulting Group, 575 Swedesford Rd. Suite 200, Wayne, PA 19087, U.S.A.*

SUMMARY

This paper presents a quantitative comparison of perfective and corrective software maintenance performed by a large military contractor using a formal program release process. The analysis techniques used in the comparison make use of basic data collected throughout the maintenance process. The data collected allow the impact of performing perfective and corrective maintenance to be quantitatively compared. Both parametric and non-parametric statistical techniques are applied to test relationships between and among process and product data. The results provide valuable information for predicting future process and product characteristics, assessing perfective and corrective maintenance impact, and quantitatively comparing the impact of both types of requirements volatility. The results also support one common rule of thumb, cast some doubt on another, and lead to the formulation of a new one.

J. Softw. Maint., **9**, 281–297 (1997)

No. of Figures: 1. No. of Tables: 12. No. of References: 7.

KEY WORDS: productivity; software process assessment; requirements volatility; maintenance process; rules of thumb; software defects

1. INTRODUCTION

Most large software systems have long lifetimes during which the software undergoes significant change. Software maintenance is defined as the set of activities performed to change a software product after the software product is delivered to the customer (Pressman, 1987, p. 25). These activities, plus the tools and methods used to maintain software, are referred to as the maintenance process. The maintenance process includes adding functionality to the software, correcting defects discovered in the software system, adapting the software to changes in the environment, and changing the software to support future maintenance or operation. The variety of changes made to software and the fact that most maintenance personnel are not involved in the development effort add significantly to the difficulties encountered while performing software maintenance.

As stated above, software maintenance is performed for a variety of reasons. Many

* Correspondence to: Professor Joel Henry, East Tennessee State University, Johnson City, TN 37614, U.S.A.
E-mail: henryj@etsu.etsu-tn.edu

CCC 1040–550X/97/050281–17$17.50

Received 9 September 1996
Revised 10 March 1997

ways exist of classifying software maintenance, such as Swanson's (1976). In contrast to Swanson's classification, the two classifications of software maintenance focused on in this study are:

1. corrective—changes made to correct defects in software; and
2. perfective—enhancements to software which provide additional functionality or enhance existing functionality.

The tasks employed during software maintenance are often similar to those applied during development. In abbreviated form and in contrast to a full software maintenance life cycle (Chapin, 1988), we focus here on the following tasks: specify, design, code and test. Thus, the first step in software maintenance is to obtain a written specification of the functionality to be added or changed. The written specification is frequently documented as changes and additions to the existing software specification. In principle the written specification is given completely and is never changed during the ensuing maintenance effort. In practice, however, these specifications are corrected and refined throughout the maintenance process. The changing of functional specifications during maintenance and development is referred to as requirements volatility. Requirements volatility has been cited as the leading problem in a field study of software managers (Thayer, Pyster and Wood, 1982). Changing requirements adversely affects the design, coding and testing of software. An acute need exists to assess quantitatively the maintenance process and the impact of requirements volatility on both the maintenance process and the software product.

The focus of this paper is the quantitative analysis of perfective and corrective maintenance activities driven by changes to the specification documents of existing software. Specifically, the analysis presented here attempts to answer three general questions:

1. What quantitative similarities exist between perfective and corrective maintenance activities?
2. What quantitative differences exist between perfective and corrective maintenance activities?
3. What do these similarities and differences suggest about the nature of perfective and corrective maintenance?

These questions are answered using a process assessment methodology (Henry and Henry, 1993) integrating the principles of Total Quality Management and the Process Maturity Framework developed at the Software Engineering Institute (Humphrey and Sweet, 1987). The assessment methodology includes the collection of process and product data, and their analysis using parametric and non-parametric statistical techniques.

While this paper describes the results obtained within a single large organization, the analysis techniques employed and the results presented may be used by other organizations to gain more visibility into the characteristics and nature of software maintenance. The statistical techniques elucidate those relationships when applied to data describing the actual maintenance processes and software.

The remainder of this paper is divided into five sections. Section 2 describes the software process used by the client to maintain the software product. Section 3 presents the methods applied for gathering and storing the data. Section 4 describes the statistical analysis techniques used on the data. Section 5 presents analysis results in five distinct areas. Section 6 outlines conclusions and the direction of future work.

2. PROCESS OVERVIEW

The Computer Program Projects Group within Lockheed-Martin manages the maintenance of a large radar-based weapons system for the U.S. Navy. The system consists of both hardware and software and enjoys a reputation of successful operation. The weapons system is controlled by more than ten different computer programs executing on separate hardware systems. The systems accept input from external sensors and operators. Extensive communication between the programs via a high speed network takes place during operation. The weapons system has undergone extensive maintenance during its more than 20 year lifetime. The maintenance efforts are largely aimed at incorporating new functionality and correcting defects.

The computer programs comprising the radar-based weapons system are released in baselines. A baseline is implemented under a contract which specifies a set of functional enhancements. Each enhancement is referred to as an upgrade item. Upgrade items typically require changes to the program performance specification (PPS), the interface design specification (IDS) or both. A baseline also includes a set of corrective items (defects) to be fixed. The process used to create new baselines is shown in Figure 1.

System personnel analyse the impact and interpret the precise meaning of each upgrade and corrective item during the system specification and software specification phases, depicted together as specification analysis in Figure 1. This analysis results in and is documented as a specification in the new PPS and the new IDS, which are reviewed at the preliminary design review (PDR). The new PPS and new IDS are the basis for implementing the new baseline. Hereafter we will simply say PPS and IDS to refer to the new versions of these documents.

Detailed design of upgrade and corrective items takes place following the PDR. Detailed analysis and design of the functionality documented in the PPS and IDS typically require additions and changes to the PPS and the IDS. The IDS and PPS pages requiring change and the detailed design are reviewed at the critical design review (CDR). Successful completion of the CDR, as judged by the customer representatives and management, initiates the implementation phase.

The new versions of the PPS and the IDS presented at the PDR and the CDR are modified at the PDR, the CDR and post-CDR. These specification changes are documented using a specification change form and implemented following the CDR. Specification changes are the driving force behind each baseline. Specification changes are a serious management concern and a major source of project difficulty in producing a new baseline because:

- the majority of specification changes occur after the CDR;
- most specification changes require code changes;

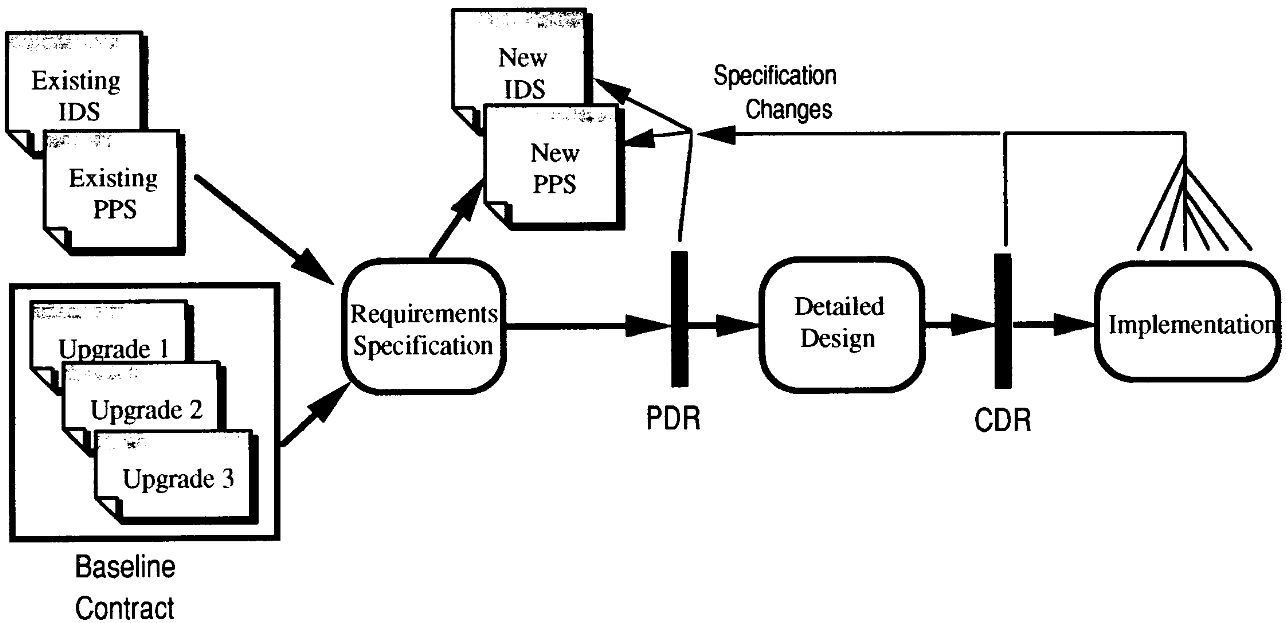


Figure 1. Baseline development process

- implementation requires co-ordination of system personnel, project managers, subcontractors, testers and technical writers; and
- the timing of the specification changes requires a new or changed functionality to be fitted into an already designed and implemented product.

The PPS and the IDS are written, maintained and controlled by the system engineering group. This group determines the need for a specification change and originates a change using a specification change form. The completed specification change form has attached to it copies of the PPS or IDS pages requiring change. The specification change form and attached pages are reviewed and approved by both management and the customer before implementation is begun.

Specification changes submitted at the PDR and the CDR are evaluated and directed during these reviews. A specification change may be directed for implementation (approved), assigned to a future baseline (deferred) or rejected (withdrawn). Specification changes approved at the PDR and/or the CDR are implemented following the CDR.

Specification changes originating after the CDR are submitted to a review board. The board consists of representatives of project management, system engineering, configuration management and the customer. A specification change may be approved, withdrawn or deferred to another baseline at the direction of the board. The board is supplied with estimates of cost and schedule impact as well as the estimated impact on computer programs.

3. DATA COLLECTION AND STORAGE

3.1. Data gathered

A large amount of data about corrective and perfective activities impacting the largest computer program in the radar based weapons system, the control program, were captured during a specific baseline. The data tracked corrective and perfective items (originating with the baseline contract) and specification changes (originating during the maintenance process). Data collected included the effort required for change (Person-days), size of change (SLOC—i.e., source lines of code—added or modified), impact of change (number of modules changed), and number of defects per change (pre- and post-delivery).

Data were collected during the baseline effort by the project management organization, the software subcontractor, the metrics group, testing groups and upper level management. Validation was done by cross-referencing data, performing an automated analysis of the delivered source code and interviewing personnel.

Specification changes approved for implementation at the PDR, the CDR and by the review board were assigned a computer program change number. This number was entered as a comment on each line of code added or changed to implement the associated specification change. In addition, the programmers implementing specification changes recorded the number of lines of code added, deleted and changed. The specification change and computer program change numbers provide traceability of each specification change. Detailed information about implementation effort, test results, and final status was also recorded for each specification change.

Table 1. Data sample sizes

Items	Sample size (number of data points)
Perfective contract items	13
Corrective contract items	18
Perfective specification changes	110
Corrective specification changes	59
Pre-delivery defects	>400*
Post-delivery defects	>100*
Modules	56
Groups	12

*Actual numbers cannot be given.

3.2. Product measures

The maintenance process includes both deliverable and non-deliverable work products. Changes to these work products, and the status of each change, must be tracked and traced in order to manage effectively the maintenance process. The numerous types of work products, as well as the volume of work products generated during a baseline require a tracking methodology supported by software tools.

Product measures at the baseline level focus on product characteristics with high visibility to the customer. These measures include the number of upgrades to be implemented, the number of defects detected, and the number of uncorrected errors along with their operational priority for each element (component) program.

Product measures at the element level capture more detailed data by element for a baseline effort. Each upgrade item receives a unique title and acronym. As the baseline progresses, specification changes are written correcting or clarifying the upgrade items. Each specification change receives a unique number and is associated with the single upgrade item it changes. Product measures then focus on each upgrade item and the associated specification changes, including number of individual requirements, number of specification changes per upgrade item, number of written requests for clarification of functionality and magnitude of impact on the element computer program.

The finest level of measurement involves detailed and specific product data. Upgrade items and specification changes approved for implementation are assigned a single distinct computer program change number. The computer program change number traces the magnitude of the impact of implementing upgrade items and specification changes on the computer programs. For each upgrade item and each specification change, both the estimate and the actual impact on each module in the system is recorded. The number of SLOC added, changed and deleted per upgrade item, specification change and module are collected. Additional measures such as the change in module size, the percentage of the module altered and the mean number of modules changed per specification change are computed from these basic measures.

The fine grain measures also track and trace defects. Each observed defect is assigned a unique defect number which is used to track subsequent review and correction activities.

A review board determines if each defect is truly a defect and approves correction of a defect for a specific baseline. If a defect is approved for correction, a computer program change number is associated with the defect. The computer program change number is used to track and trace error correction impact on the product.

Following delivery of a baseline, the customer assumes responsibility for installing the software system in a chosen ship or ship class, and for maintaining the computer programs until delivery of the next baseline. The customer captures maintenance data about delivered computer programs including number and type of defects, and the impact of error correction on the computer programs. This single, consistent measurement approach allowing detailed statistical analysis to be performed benefits both the maintenance organization and the customer.

3.3. Process measures

Process measures are collected from the same three levels of detail used to collect product data. Process measures consistently focus on effort expended, process effectiveness, and resource allocation throughout all three levels. The measures provide increasingly more detailed data about process tasks and activities as one moves down through the process tiers.

Baseline process measures include effort expended, progress toward baseline milestones and task completion rates per element. Effort expended per element provides insight into cost and schedule status, and provides feedback on the accuracy of initial effort estimates. Progress measures assess the status of each element within each major process phase. Task completion rates, used in conjunction with progress measures can be used to replan process phase completion dates. These process measures provide important information to upper level management without overwhelming them with detailed metric data.

Element level process measures include effort expended, progress and task completion status, but focus on upgrade items, specification changes and error corrections within the element. For example, the effort expended to implement each upgrade item is captured. Progress is monitored by recording the number of requirements designed, implemented, integrated and tested per upgrade item. Task completion measures monitor the number of unit tests completed per week, the number of defects corrected per week and the specification changes written per month.

The most specific level of process measurement includes effort expended per activity, work product and progress. Effort expended measures capture data per activity such as unit test time, and per work product, such as number of days spent correcting each software module for each error corrected. Progress measures quantify fine grain activities such as the total number of tests run per specification change.

The measurement approach results in the collection of a large amount of metrics data related in many important ways. Measures are carefully collected, validated and stored in a relational database. The collection approach requires automation to reduce the impact of measurement on the maintenance staff. While the collection and storage of measures are extremely important, the procedures required to capture measures cannot overburden the maintenance team and prevent them from building high quality products on time and within budget.

4. ANALYSIS TECHNIQUES

The analysis techniques presented in this paper use both parametric and non-parametric statistics. Parametric statistical techniques assume data are drawn from a population adhering to a specific type of distribution (e.g., normal). Non-parametric techniques do not assume any distribution and are valid for all populations (Ott, 1988, p. 183).

The association between data values is of primary interest. Association is analysed using linear and rank correlation. Linear correlation measures the linear relationship between two variables. Rank correlation does not assume a linear relationship but rather tests the relationship based on ranks assigned to data values. The rank of a data value is the position of the data value in an ordered list of all data values. In the case of both linear and rank correlation, correlation values range from $-1{\cdot}0$ to $+1{\cdot}0$. Correlation approaching $-1{\cdot}0$ demonstrates a strong inverse relation between variables, meaning an increase in one variable is associated with a decrease in the other. Correlation approaching $+1{\cdot}0$ demonstrates a strong positive relation between variables, meaning just the opposite of an inverse relation: an increase in one variable is associated with an increase in the other.

Another goal of analysis is to compare the means of samples drawn from two or more independent populations to detect population differences. The Mann–Whitney rank sum test and the Kruskal–Wallis test compare means of samples taken from two or more populations (Ott, 1988, pp. 183–188). Both tests produce a *P*-value, which is a probability value ranging from zero to one. Both tests offer an answer to the question:

If the populations really have the same mean overall, what is the probability that random sampling would reveal a difference between sample means as large (or larger) than observed?

The test hypothesis for sample mean comparisons is that the sample means are equal. A small *P*-value indicates that the small probability means of the two samples are in fact similar and we can reject the test hypothesis with a small probability of error. A large *P*-value indicates rejecting the test hypothesis has a high probability of error. In short, there is not enough evidence to reject the test hypothesis that the populations have identical means.

A simple classification of software products based on data available early in the maintenance process which predicts future product characteristics is also valuable. For example, classifying software components into categories of error-proneness based on their size is an important classification scheme if a statistically significant association exists between size and number of defects per component. Contingency tables are used to measure the statistical significance of such associations.

In a contingency table the rows classify items (e.g., components) using one criteria (e.g., size) and the columns classify items using a second criteria (e.g., number of defects). Contingency tables can be used to test the hypothesis that row classification is independent of column classification. This hypothesis is tested using a *P*-value derived from the classification of actual data. The *P*-value indicates the likelihood that row and column classifications are independent. A small *P*-value causes the test hypothesis to be rejected in favour of assuming row classification and column classification are dependent.

In our example, a small P-value would indicate that large components should be classified as error-prone.

The statistical techniques described here are basic non-parametric tests available in most commercially available statistical software packages. The results of these statistical tests are easily interpreted and presented. Our experience has shown that simple statistical techniques applied to basic data are much more useful in the commercial world than elegant techniques applied to complex data.

5. ANALYSIS RESULTS

5.1. Analysis guidelines

Four significant results are described in Subsections 5.2 and 5.3. Each subsection discusses the focus of the analysis, the data used in the analysis, and the statistical results. A maximum P-value of 0·05 and the minimum R^2 value of 0·80 were established as criteria for asserting that relationships existed. This maximum P-value of 0·05 represents a five per cent chance of mistakenly assuming a relationship exists. The minimum R^2 can be viewed as explaining 80 per cent of the variability of the predicted variable.

5.2. Corrective and perfective maintenance similarities

5.2.1. Findings

In this section we examine the data for similarities between corrective and perfective maintenance activities. In order to compare both types of maintenance, we analysed corrective and perfective activity measures using basic statistical techniques. After examining the basic statistical results, we found similarities in productivity for both types of maintenance. In this section we describe and interpret these statistically significant similarities.

5.2.2. Corrective items versus perfective items

An important area of interest in software maintenance is productivity. This area is particularly of interest because corrective and perfective maintenance activities are often considered very different types of tasks. We began our investigation by examining productivity for corrective and perfective maintenance items (specified in the baseline contract). We measured productivity in source lines of code (SLOC) added, modified and deleted per person-day, including the time spent in specification, design, coding and testing.

Examining Table 2, little difference appears to exist between corrective and perfective productivity, measured by SLOC per person-day. The means (1·8996, 1·7924) and medians (1·4032, 1·7274) for both types of maintenance are nearly the same. However, the standard deviations are different, implying wider variances for corrective maintenance.

Table 3 shows strong linear correlations between change per module (SLOC per module) and productivity (SLOC per person-day). Both corrective and perfective productivity have

Table 2. Basic corrective and perfective maintenance statistics

Measure	Corrective (SLOC per person-day)	Perfective (SLOC per person-day)
Mean	1·8996	1·7924
Standard deviation	2·4980	1·0323
Median	1·4032	1·7274
Minimum	0·1667	0·4865
Maximum	10·7190	4·1333

Table 3. Linear correlations of product impact versus corrective and perfective productivity

Item	Corrective item (SLOC per person-day)	Perfective item (SLOC per person-day)
Total SLOC	+0·409	+0·793
SLOC per module	+0·951	+0·788

high correlations (+0·951, +0·788) with change per module. Table 3 suggests that as SLOC changed per module increases, productivity also increases.

Strong linear correlations between SLOC changed per module and productivity can be very useful in planning software maintenance efforts, evaluating software maintenance costs, and allocating resources. For example, our results suggest that a maintenance item impacting 100 SLOC in one module will provide higher productivity than a maintenance item impacting 10 SLOC in each of ten different modules.

Tables 2 and 3 suggest that no great difference exists between corrective-item mean productivity and perfective-item mean productivity. In order to evaluate the statistical significance of the difference in means between corrective and perfective maintenance item productivity, we applied the Mann–Whitney test. A *P*-value of 0·9833 is observed which is greater than the previously established maximum *P*-value of 0·05. The *P*-value of 0·9833 is not an adequate basis to reject the test hypothesis that the productivity achieved performing corrective and perfective maintenance is similar.

These results indicate that average productivity for both types of maintenance does not differ significantly. Both types of productivity appear to be influenced by the distribution of change across the software product. However, the variation in productivity does appear to differ, with corrective maintenance having a larger standard deviation. We believe this results from the difficulty in finding and correcting errors hidden in the software product while perfective maintenance begins from a more stable requirements definition.

5.2.3. Corrective SCs versus perfective SCs

In the previous subsection we were unable to reject the hypothesis that the productivity of corrective items and perfective items is statistically different. In our study, item level

Table 4. Basic statistics for productivity of corrective and perfective SCs

Measure	Corrective (SC SLOC per person-day)	Perfective (SC SLOC per person-day)
Mean	1·5833	1·4489
Standard deviation	1·9887	1·4979
Median	1·0000	1·0581
Minimum	0·0000	0·0000
Maximum	10·7190	8·6000

analysis does not include requirements volatility, which we refer to as specification changes (SCs), We were able to collect data for corrective and perfective specification changes separately. Again, the investigative method begins with examination of the basic statistics, followed by consideration of correlations, and finally application of the Mann–Whitney test. Results are then interpreted.

Table 4 shows little difference between corrective SC and perfective SC characteristics. In both cases, the means (1·5833, 1·4489) and the standard deviations (1·9887, 1·4979) are quite similar for productivity. However, basic statistics serve only as a starting point in determining similarities and differences; additional analysis must be performed in order to compare statistically the two data sets.

Table 5 contains the linear and rank correlations between productivity and changed SLOC per module. The linear correlations for corrective SCs and perfective SCs do not express great similarities. To account for outlying data points, we examined rank correlation. Rank correlation uses the rank of each data item in an ordered list of data items rather than the value when evaluating correlation. This ranking minimizes the impact of outliers on correlation. The rank correlations are virtually the same, implying that SC productivities correlate to product impact in the same way item productivities correlate to product impact.

In order to evaluate statistically the differences between productivities, the Mann–Whitney test is again applied. The P-value produced is 0·4959, meaning that there is a 49·59 per cent chance of error in rejecting the test hypothesis. This P-value is greater than 0·05 which means we cannot reject the hypothesis that productivities of the corrective SCs and the perfective SCs shown in Table 4 are similar.

Table 5. Correlations between maintenance productivity and SLOC/module

Correlation	Corrective (SLOC per person-day)	Perfective (SLOC per person-day)
Linear correlation of SLOC per module	+0·9210	+0·5370
Rank correlation of SLOC per module	+0·8996	+0·9076

Table 6. Basic statistics for corrective SC and perfective SC characteristics

Measure	Total SLOC		Modules changed		Person days	
	Corrective	Perfective	Corrective	Perfective	Corrective	Perfective
Mean	33·1905	150·8511	1·7541	3·0459	18·3984	60·9073
Standard deviation	55·3804	517·6439	1·7763	3·6676	20·9726	133·9737
Median	10·5000	23·500	1·0000	2·0000	7·5000	16·0000
Minimum	0·0000	0·0000	0·0000	0·0000	0·0000	0·0000
Maximum	253·0000	4 579·0000	9·0000	21·0000	91·0000	952·5000

Similarities in average productivity, standard deviation and the Mann–Whitney test suggest no difference in productivity between corrective and perfective specification changes. We believe this results from the well specified process for reviewing, approving and implementing all specification changes within the overall maintenance process. This stable specification change process produces similar productivity.

5.3. Corrective and perfective maintenance differences

5.3.1. Process and product impact of requirements volatility

Subsection 5.1 suggests corrective and perfective productivity are similar for both contract items and specification changes. In this subsection we present the significant differences discovered between corrective and perfective specification change activity.

Specification changes document and track requirements volatility in our maintenance process. Data collected for each specification change allowed us to examine the impact of requirements volatility. We specifically wanted to know if corrective and perfective specification changes require similar effort (measured by total person-days expended), and result in the similar product impact (measured by size of change and number of modules changed). To answer this question, we examined basic measures of each type of maintenance, and then applied the Mann–Whitney test to assess the differences.

Table 6 shows the basic statistics for corrective specification changes and perfective specification changes. Comparing corrective and perfective specification change impact, we observe differences in two characteristics. Specifically, the means and standard deviations appear different. To evaluate these differences, we apply the Mann–Whitney test to size of change, number of modules changed and person-days expended for both perfective and corrective specification changes.

Table 7 contains the results of the Mann–Whitney tests for effort, modules changed,

Table 7. Mann–Whitney test P-values for corrective and perfective SC characteristics

Metric	Total SLOC	Modules changed	Person-days
P-value	0·0041	0·0170	0·0012

and size. Notice that the *P*-values are all less than the maximum *P*-value of 0·05. The *P*-values in Table 7 allow us to reject the test hypotheses that effort, modules changed, and size of change for corrective specification changes are the same as corresponding measures of perfective specification changes. Thus, we can conclude that the amount of effort, the number of modules and the number of lines of code are significantly different for perfective specification changes than for corrective specification changes in our study.

Our data indicate that these measures are significantly larger for perfective specification changes than for corrective specification changes. The Mann–Whitney test we employed identified only that the data for these two types of requirements volatility are different, and did not identify the direction of that difference. A more controlled experiment is needed to evaluate a directional hypothesis.

Perfective requirements volatility activities do not require similar amounts of effort and do not result in similar amounts of change to software products, compared with corrective requirements volatility. We believe these differences are due to the nature of the changes. Perfective specification changes typically result in expansion of system requirements while corrective specification changes represent correction of errors in the specification document. While some exceptions occur, this difference in the nature of the change results in a difference in effort and product impact.

5.3.2. Impact of changed SLOC on number of errors

This subsection considers the impact of corrective and perfective activities on software quality. While requirements volatility affect the process and the software product differently, what is the effect on overall quality? In this subsection we examine the impact of corrective and perfective maintenance on the software product prior to delivery and following delivery to the customer.

Prior to delivery, maintenance impact and quality (measured by defects per module) can be assessed for each software module, as shown in Table 8. Following delivery, impact and quality (measured by defects per group) can be assessed by groups of modules. Each group is a unique, non-overlapping set of modules which provides specific functionality. Table 8 contains module measures (pre-delivery) and Table 9 contains group measures (post-delivery).

The basic statistics, presented in Tables 8 and 9, suggest a difference in the number of changed SLOC resulting from implementation of perfective specification changes and

Table 8. Module measures of changed SLOC and number of defects

Measures per module	Pre-delivery defects	Corrective changed SLOC	Perfective changed SLOC
Mean	10·6071	226·0370	619·1071
Standard deviation	11·2115	706·8587	1 246·7687
Median	8·5000	49·0000	129·0000
Minimum	0·0000	0·0000	0·0000
Maximum	57·0000	3 718·0000	6 518·0000

Table 9. Module group measures of changed SLOC and number of defects

Measures per group	Post-delivery defects	Corrective changed SLOC	Perfective changed SLOC
Mean	14·4545	871·8571	2 800·7500
Standard deviation	13·6041	1 263·0267	4 033·2775
Median	8·0000	373·0000	1 020·5000
Minimum	1·0000	172·0000	0·0000
Maximum	41·0000	3 718·0000	11 216·0000

Table 10. Rank correlations between quality and changed SLOC

Defects	Corrective changed SLOC	Perfective changed SLOC
Pre-delivery defects	+0·3215	+0·9702
Post-delivery defects	+0·2143	+0·8884

corrective specification changes. However, the question is, is this difference statistically significant?

To begin evaluating, we start by looking at the rank correlation between quality and size of change (we used rank correlation due to outlying data points in our data sample). Table 10 presents the rank correlations between changed SLOC for corrective specification changes and perfective specification changes, and the number of pre-delivery and post-delivery defects detected. The number of perfective changed SLOC has a much stronger positive correlation to both types of defects than the number of corrective changed SLOC. These results suggest that as the number of perfective changed SLOC increases, the number of pre-delivery and post-delivery defects also increases. Conversely, corrective changed SLOC appears to have little relationship to pre-delivery or post-delivery defects. To further analyse the difference we have found, we have constructed contingency tables, Tables 11 and 12.

Table 11 shows no clear relationships exist between the number of corrective changed SLOC and the number of pre-delivery defects. (Note: only 27 out of 56 modules were affected by corrective maintenance activities). Statistically, the table supports the hypothesis

Table 11. Number of pre-delivery defects versus corrective changed SLOC

Defects	⩽ Median number of corrective changed SLOC	> Median number of corrective changed SLOC
⩽ Median number of pre-delivery defects	10	12
> Median number of pre-delivery defects	3	2

Table 12. Number of pre-delivery defects versus perfective changed SLOC

Defects	⩽ Median number of perfective changed SLOC	> Median number of perfective changed SLOC
⩽ Median number of pre-delivery defects	24	8
> Median number of pre-delivery defects	4	20

that the number of corrective changed SLOC and the number of pre-delivery defects are independent.

However, Table 12 shows a strong relationship exists between perfective changed SLOC and the number of pre-delivery defects. Specifically, Table 12 reveals that when perfective specification changes require more than the median number of SLOC to be changed, the result is more than the median number of defects in 20 out of 28 times (see the second column of Table 12). A statistical test of Table 12 produces a chi-square value of 18·667 with one degree of freedom, meaning the probability that the observed results occurred by chance is less than 0·001 (less than one-tenth of one per cent and less than 0·05).

The strong results obtained from the rank correlations and contingency tables suggest that requirements volatility during perfective maintenance is more likely to result in more pre-delivery defects than requirements volatility during corrective maintenance. We believe these results occur because perfective maintenance results in more product impact and much of this impact is new code. Corrective maintenance does not impact the product as significantly as perfective maintenance. Corrective maintenance also removes a known defect and, in our case, does not appear to introduce significant amounts of new defects, as is often described in the software engineering literature (Pressman, 1987, p. 189). Perfective maintenance does appear to introduce new defects, which is typical in new code.

6. CONCLUSIONS

The results of this investigation reveal several interesting characteristics of software maintenance. Viewing the similarities, differences and statistical relationships between perfective and corrective maintenance confirms a previously advanced ‘rule of thumb’, questions another such rule and leads to the proposal of a new rule.

Analysis of perfective and corrective maintenance items, the highest level view of maintenance analysed, showed no statistical difference between corrective and perfective productivity. The baseline contract includes both types of items, indicating these items are specified in the initial stages of the maintenance effort. Given that both perfective and corrective items are specified early and are implemented using an identical process, it is not surprising that no significant statistical difference exists in productivity. Similarly, corrective and perfective productivity during requirements volatility did not differ statistically.

Further examination showed productivity for both types of maintenance to be positively correlated with changed SLOC per module. The distribution of change appears to affect productivity more than the size of the change. While the size and distribution of

change to the product differed significantly between perfective and corrective maintenance specification changes, productivity did not. Perfective specification changes resulted in larger and more distributed change to the software product than did corrective specification changes. However, productivity did not show a significant statistical difference because the average change per software module remained roughly the same for both types of specification changes. These results confirm the old rule of thumb:

the more local the change to the software product, the easier the maintenance effort.

Analysis of the impact of perfective and corrective maintenance on the quality of the delivered software product provided two interesting results. First, strong positive rank correlation existed between the impact of perfective maintenance and the number of post-delivery defects detected in the software. This correlation suggests that as the impact of perfective maintenance increases, the number of post-delivery defects also increases. Second, a positive but not significant correlation exists between the impact of corrective maintenance and the number of post-delivery defects. This result raises doubt about the old rule of thumb:

fixing errors inserts new errors into a software product.

Our results suggest a new rule of thumb:

with a defined maintenance process in place, increasing corrections to requirements during maintenance has little impact on post-delivery defects.

Obviously a realistic limit to this rule exists. The number of post-delivery defects could not be held constant indefinitely while the amount of corrective maintenance increased. However, our results show long-term corrective maintenance which affects both requirements and code does not significantly decrease software quality.

These results illustrate two final points. First, neither the size of the change nor the distribution of the change, taken individually, influence productivity. It is the combination of these factors which significantly impact the productivity of both perfective and corrective maintenance activities. Second, perfective and corrective requirements volatility differ significantly in both their impact on the software product and their impact on post-delivery defects.

Our results can be used in several important ways. Project managers can use our results for more accurate planning. For example, a requirements change affecting many modules across a software product will be less productive to implement. Our results also suggest a wider variance in corrective maintenance productivity. New functionality added to a software system will contain more defects. Project managers should plan for more thorough design and code reviews, and additional testing on the new functionality. For a legacy system, defect correction does not appear to increase system defects significantly.

Finally, organizations can follow our tracking and tracing methods for collecting maintenance data, then apply statistical tests to determine the specific product and process

relationships. These relationships provide quantitative insight into software maintenance and can help improve an organization's maintenance processes and the resulting software.

References

Chapin, N. (1988) 'Software maintenance life cycle', in *Proceedings Conference on Software Maintenance—1988*, IEEE Computer Society Press, Los Alamitos, CA, pp. 6–13.

Henry, J. E. and Henry, S. (1993) 'Quantitative assessment of the software maintenance process and requirements volatility', in *Proceedings CSC '93*, ACM Press, New York, NY, pp. 346–351.

Humphrey, W. S. and Sweet, W. L. (1987) *A Method for Assessing the Software Engineering Capability of Contractors*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 116 pp.

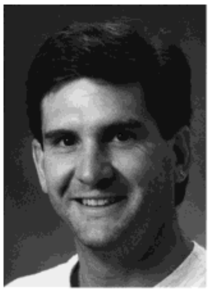
Ott, L. (1988) *Introduction to Statistical Methods and Data Analysis*, PWS-KENT Publishing Co., Boston, MA, 835 pp.

Pressman, R. S. (1987) *Software Engineering: A Practitioners Approach*, McGraw-Hill Book Co., New York, NY, 852 pp.

Swanson, E. B. (1976) 'The dimensions of maintenance', in *Proceedings of the 2nd International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 492–497.

Thayer, R. H., Pyster, P. and Wood, R. C. (1982) 'Validating solutions to major problems in software engineering project management', *Computer*, **15**(8), 65–77.

Authors' biographies:



Joel Henry received his Ph.D. in computer science from Virginia Tech in 1993, and both the Bachelor of Science and Master of Science degrees in Computer Science from Montana State University (Bozeman, MT) in 1985 and 1986, respectively. Currently he is Assistant Professor at East Tennessee State University (Johnson City, TN). Dr. Henry consults extensively for software development organizations across the U.S., and conducts research on software process assessment and improvement, object-oriented development, and software quality.

James P. Cain received his Bachelor of Science degree in mathematics from Dickinson College (Carlisle, PA) in 1994. In 1996, he earned a Master of Science degree in computer science with a concentration in software engineering from East Tennessee State University (Johnson City, TN). He is currently a consultant for Integrated Systems Consulting Group in Wayne, PA where he develops client-server document management solutions for Fortune 500 companies.